REMARKS/ARGUMENTS

Claims 1-22 are pending in the present application. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 102, Asserted Anticipation

The Examiner rejects claims 1-22 as anticipated by *Crow et al.*, Versatile Indirection In an Extent Based File System, U.S. Patent Application Publication 2004/0254907 (December 16, 2004) (hereinafter "*Crow*"). This rejection is respectfully traversed. In regards to claim 1, the Examiner asserts:

> *Crow* discloses in figure 8C, a method in a data processing system for storing data in a file system, the method comprising determining whether space is available in an inode for a file in the file system; and responsive to space being available, storing the data in the inode (paragraph [0048], [0053]).

Office Action dated April 27, 2006, p. 2.

A prior art reference anticipates the claimed invention under 35 U.S.C. §102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Crow does not anticipate claim 1 because *Crow* does not teach all the features of claim 1. Claim 1 is as follows:

> 1. A method in a data processing system for storing data in a file system, the method comprising:
>
> determining whether space is available in an inode for a file in the file system; and
>
> responsive to space being available, storing the data in the inode.

Crow does not anticipate claim 1 because *Crow* does not teach the feature of, "determining whether space is available in an inode *for a file in the file system* and responsive to space being available, storing the *data* in the inode" as recited in claim 1. The Examiner asserts otherwise, citing *Crow*'s figure 8C, reproduced below:

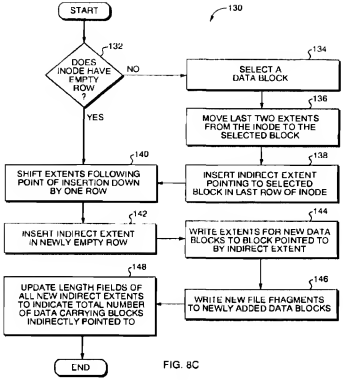


FIG. 8C

According to figure 8C, in step 132, "the operating system first determines whether at least one empty row remains for writing a new *extent* to the file's inode." *Crow*, paragraph 41 (emphasis added). In step 146, the new *file fragments are stored in newly added data block* and not in the inode. The inode stores the extent which contains a pointer that indicates both the logical volume and a physical offset of the data block. *Crow*, paragraphs 33-34. Therefore, figure 8C does not teach the features of claim 1 because figure 8C determines whether there is space in the inode for an "extent" and not for a file *in the file system* as recited in claim 1. Furthermore, figure 8C discloses storing the data in a newly added data block instead of storing the data in the inode as recited in claim 1.

Additionally, the Examiner cites to the following portions of *Crow* as disclosing the features of claim 1:

> [0048] Later, a request from a software application for more *data blocks for the file* is received by the operating system (step 157). In response to the request, the operating system determines whether the region contiguous to the physical location of the previous segment of the file has more available data blocks (step 158). If region has more available blocks, the operating system allocates a new string of blocks immediately following the physical location previous segment, i.e., contiguous with the previous segment (step 160). Then, the operating system increases the value of the length stored in the length field of the previous extent for the region by the number of blocks in the new string (step 161). If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162). The newly found logical volume may be a

different logical volume. Thus, the new string of data blocks may be allocated to the file from a different logical volume.

[0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraphs 48 and 53.

Neither the cited portion nor any other portion of *Crow* teaches the feature of, "determining whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the *data* in the inode," as recited in claim 1. Paragraph 48 only discusses an application requesting more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in paragraph 48 discloses the features of claim 1. Therefore paragraph 48 is completely irrelevant to the claimed feature, "determining whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the *data* in the inode," as recited in claim 1.

Similarly, paragraph 53 discloses the use of an inode to store a data file. However, nothing in paragraph 53 or any other portion of *Crow* teaches the precondition to storing the data in the in the inode as recited in claim 1. In other words, *Crow* does not teach, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available," as recited in claim 1. *Crow* does not perform a pre-determination prior to storing the data in the inode. *Crow* performs a post-determination. *Crow* performs the function of storing the data in the inode and "when the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents." *Crow*, paragraph 53. Therefore, this portion of *Crow* does not teach, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the data in the inode," as recited in claim 1.

Because *Crow* does not teach all of the features of claim1, *Crow* does not anticipate claim 1. Additionally, claims 8, 9 and 16 claim similar features to claim 1. Therefore, the same distinctions between *Crow* vis-à-vis claim 1 apply to these claims. Accordingly, the rejection of claims 1, 8, 9, and 16 has been overcome.

Regarding claim 2, the Examiner asserts:

Crow discloses in figures 8A-8C1 and 10, to determining whether additional data being present; and responsive to the additional data being present, storing the additional data in a partially filled block of another file (paragraph [0038], [0039], [0042] and [0044]).

Office Action dated April 27, 2006, p. 2.

Crow does not anticipate claim 2 because *Crow* does not teach all the features of claim 2. Claim 2 is as follows:

> 2. The method of claim 1 further comprising:
> determining whether additional data is present; and
> responsive to the additional data being present, storing the additional data in a partially filled block of another file.

Because claim 2 depends from claim 1, the same distinctions between *Crow* and claim 1 apply to claim 2. Additionally, claim 2 claims other additional combinations of features not suggested by the reference. Specifically, *Crow* does not teach the feature of, "storing *the additional* data in a *partially filled* block *of another file*," as recited in claim 2. The Examiner asserts otherwise, citing the following figures:

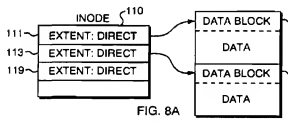


FIG. 8A

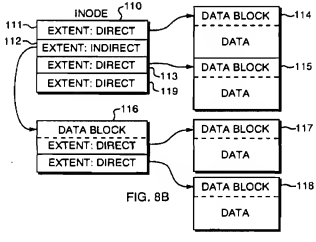
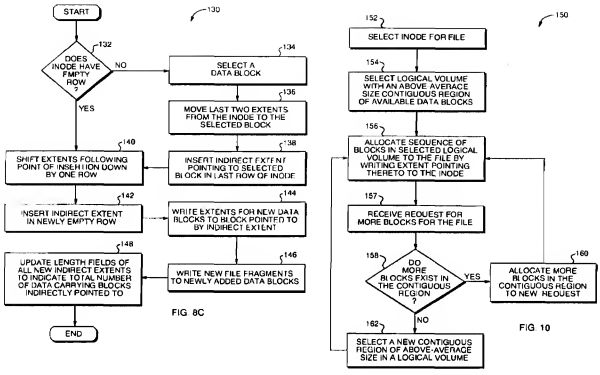


FIG. 8B



Figures 8A, 8B, and 8C do not teach the feature of, "storing *the additional* data in a *partially* filled block *of another file*." Instead, figures 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to ***new*** data block 117 and original data block 215. *Crow*, paragraph 38. This feature enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. *Crow*, paragraph 39. Figure 8C is a flow chart illustrating a method 130 of inserting a new file segment between two adjacent file segments. *Crow*, paragraph 39. In step 148, the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent. *Crow*, paragraph 44.

However, these figures do not teach "storing *the additional* data in a *partially filled* block *of another file*," as recited in claim 2. Instead, these figures show how to add ***new*** data blocks to a file using *indirect extents*. Therefore, figures 8A, 8B, and 8C do not anticipate the claimed features of claim 2.

Furthermore, figure 10 of *Crow* does not teach the feature, "storing *the additional* data in a *partially filled* block *of another file*." Figure 10 illustrates a method 150 of allocating data blocks to a file from a plurality of logical volumes. The operating system allocates a string of data blocks from the contiguous region of the selected volume to the file by writing an extent. The extent points to the string in the first row of the inode assigned to the file (step 156). Step 157 occurs when an application requests more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately

following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in Figure 10 teaches the feature, "storing *the additional* data in a *partially filled* block *of another file.*" Instead, Figure 10 teaches a method of storing data in contiguous blocks, which is different from the features of claim 2. Therefore, figure 10 does not anticipate claim 2.

Additionally, the Examiner cites to the following portions of *Crow* as disclosing the features of claim 2:

> [0038] FIGS. 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. FIG. 8A shows an inode 110 assigned to the file. The inode 110 has consecutive direct extents 111, 113, 119 that point to data blocks 114, 215, 330 storing originally consecutive segments of the file. FIG. 8B shows the final file in which an indirect extent 112 has been inserted between the two original direct extents 111, 119. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to **new** data block 117 and original data block 215. Since the indirect extent 112 is physically located between the two original extents 111, 119, the segments stored in the blocks 117, 215 (indirectly pointed to) are logically located between the original segments stored in the blocks 114, 330. Inserting the indirect extent 112 has grown the middle of the associated file by logically inserting the segment in **new** data block 117 between the originally consecutive segments in data blocks 114 and 215.

Crow, p. 3, ¶ 38 (emphasis added).

> [0039] The file system, illustrated in FIGS. 5-8B, allows any extent of an inode to be indirect, because the flag field indicates the type of each extent. This free placement of indirect extents within the inodes enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. To insert a new data segment, the system inserts an indirect extent into the file's inode between the two extents for the selected data segments. Then, the system makes the indirect extent point to a data block storing new direct extents that point, in turn, to the consecutive pieces of new data segment. The new direct extents are logically located in the inode at the point where the new indirect extent has been inserted.

Crow, p. 3, ¶ 39 (emphasis added).

> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a **new** data block pointed to by the new direct extent (step 138).

Crow, p. 3, ¶ 42 (emphasis added).

> [0044] Next, the operating system inserts an indirect extent into the row of the inode previously occupied by the extent now in the second row of the indirect

> block (step 146). The new indirect extent points to the new indirect block and has a length equal to the sum of the lengths of both extents in the indirect block. In FIG. 8B, the operating system writes the extent 112 pointing to the data block 116 to the inode 110. Finally, the operating system writes the new file segment in the **new** data block pointed to by the new direct extent (step 148). In FIG. 8B, the new file segment is written to the data block 117.

Crow, p. 3, ¶ 44 (emphasis added).

None of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Paragraphs 38 and 39 describe figures 8A and 8B. As previously discussed, this portion of *Crow* explains how to add ***new*** data blocks to a file using indirect extents. Similarly, paragraphs 42 and 44 describe figure 8C. Both paragraphs specifically state: "the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent." Therefore, none of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Accordingly, the rejection of claim 2 has been overcome.

Because the remaining claims depend from one of claims 1, 9, or 16, the same distinctions between *Crow* and claims 1, 9, or 16 apply to the remaining claims. Additionally, the remaining claims claim other additional combinations of features not suggested by the reference. For example, claims 3, 4, 7, 10-12, 15, 17-19, and 22 claim a feature containing a "partially filled block of another file." As previously discussed in claim 2, *Crow* does not teach the use of storing data in a partially filled block of another file. Therefore, *Crow* does not teach or suggest the additional features of claims 3, 4, 7, 10-12, 15, 17-19, and 22.

Furthermore, *Crow* does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. The present invention provides an improved method over *Crow*. By storing data in a partially filled block of another file as claimed, disk space utilization is improved because of the reduction of blocks with unused space (Specification p. 16). Absent the Examiner pointing out some teaching or incentive to implement *Crow* with the use of a partially filled block of another file, one of ordinary skill in the art would not be led to modify *Crow* to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify *Crow* in this manner, the presently claimed invention can be reached only through an improper use of hindsight using Applicants' disclosure as a template to make the necessary changes to reach the claimed invention. Consequently, it is respectfully urged that the rejection of the remaining claims have been overcome.

II. Conclusion

It is respectfully urged that the subject application is patentable over *Crow* and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: July 27, 2006

Respectfully submitted,

/Theodore D. Fay III/

Theodore D. Fay III
Reg. No. 48,504
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants

TF/NH